A Session with ChatGPT – PowerShell for Git

# Version Information

Satya Komatineni, 10/17/23

Contents

[Version Information 1](#_Toc148433861)

[Goal/Note 1](#_Toc148433862)

[English Spec for the program 1](#_Toc148433863)

[After a few iterations of adjusting the iterative prompts. 2](#_Toc148433864)

[What are the iterations? 4](#_Toc148433865)

[Caveats 4](#_Toc148433866)

# Goal/Note

1. Use ChatGPT for help with Programming.
2. Write a PowerShell command line program to automate a few often-used Git Commands
3. Why Powershell?
   1. PowerShell has some peculiarities
   2. I love PowerShell and wrote about 20,000 lines of code a few years ago
   3. But PowerShell initial learning is an “intricate web”, so I don’t remember much of it
   4. And hence asking ChatGPT for help
4. Why Git?
   1. Lost less so, but similar to PowerShell, for occasional users Git is a “scoundrel”
   2. I usually use VSCode plugins to over come this
   3. But sometimes “I just want to” upload the code. No fuss
   4. So finally “gave in” to command line approach “for simple tasks” and simple repos like “articles where this document is”
   5. However I wanted a helper command file that abstracts of out “git”

# English Spec for the program

1. It should allow for multiple options that it prommpts the user with a list of options

2. The options are:

1. Git init: (init and replicate a repo from github)

2. Status: (Tell me the current local repo status as to how current the local repo is compared to its remote)

3. Upload: (Push all local files to remote)

4. Exit the program

3. For options 1 and 2 provide no code but simply acknoledge with "ok"

4. For option 4, exit the program

3. For option 3 write code that does the following

1. Prompt for a commit message

2. Execute add for all files and sub directories

3. Prompt that "add" is succesful or failure and if the program should continue or not

4. If the user says "Y" or "Yes" (case insensitive) continue, otherwise terminate the program

5. If continue do a commit with the provided message

6. Display the message from commit

7. Prompt to proceed or not

8. If the user says "Y" or "Yes" (case insensitive) continue, otherwise terminate the program

9. Then push to the remoted indicated by "origin" and the branch "master"

10. Display the message from push

11. exit the program

# After a few iterations of adjusting the iterative prompts.

This code has the following characteristics:

1. It is structured using explicit functions for each job
2. The nested “if” and “else” are eliminated using a more select-and-return approach to make the code more readable.

Note: I have eyeballed the code for accuracy but have not tested yet.

Code after a few iterations follow:

1. # Function to initialize Git repository

2. function Initialize-GitRepository {

3. Write-Host "ok" # Git init

4. }

5.

6. # Function to display Git repository status

7. function Get-GitStatus {

8. Write-Host "ok" # Status

9. }

10.

11. # Function to explain the selected option

12. function Explain-Option {

13. param (

14. [int]$optionNumber

15. )

16.

17. switch ($optionNumber) {

18. 1 { Write-Host "Option 1: Git init (init and replicate a repo from github)" }

19. 2 { Write-Host "Option 2: Status (Tell me the current local repo status as to how current the local repo is compared to its remote)" }

20. 3 { Write-Host "Option 3: Upload (Push all local files to remote)" }

21. 4 { Write-Host "Option 4: Explain option (Provide an explanation for the selected option)" }

22. default { Write-Host "Invalid option number. Please enter a number between 1 and 4." }

23. }

24. }

25.

26. # Function to stage changes using 'git add'

27. function Stage-GitChanges {

28. git add -A

29. return $?

30. }

31.

32. # Function to commit changes using 'git commit'

33. function Commit-GitChanges {

34. param (

35. [string]$commitMessage

36. )

37. git commit -m $commitMessage

38. return $?

39. }

40.

41. # Function to push changes using 'git push'

42. function Push-GitChanges {

43. git push origin master

44. return $?

45. }

46.

47. # Function to upload changes to Git repository

48. function Upload-GitChanges {

49. $commitMessage = Read-Host "Enter commit message"

50. $addSuccess = Stage-GitChanges

51.

52. if (-not $addSuccess) {

53. Write-Host "Add failed. Please check your changes and try again."

54. return

55. }

56.

57. Write-Host "Add successful."

58.

59. $continueCommit = Read-Host "Continue with commit? (Y/N)"

60. if ($continueCommit -ne "Y" -and $continueCommit -ne "Yes") {

61. Write-Host "Commit operation canceled."

62. return

63. }

64.

65. $commitSuccess = Commit-GitChanges -commitMessage $commitMessage

66. if (-not $commitSuccess) {

67. Write-Host "Commit failed. Please check your changes and try again."

68. return

69. }

70.

71. Write-Host "Commit message: $commitMessage"

72.

73. $continuePush = Read-Host "Continue with push? (Y/N)"

74. if ($continuePush -ne "Y" -and $continuePush -ne "Yes") {

75. Write-Host "Push operation canceled."

76. return

77. }

78.

79. $pushSuccess = Push-GitChanges

80. if (-not $pushSuccess) {

81. Write-Host "Push failed. Please check your network connection and try again."

82. return

83. }

84.

85. Write-Host "Push successful."

86. }

87.

88. # Main function to display menu and handle user input

89. function Main {

90. do {

91. Write-Host "Options:"

92. Write-Host "1. Git init"

93. Write-Host "2. Status"

94. Write-Host "3. Upload"

95. Write-Host "4. Explain option"

96. Write-Host "5. Exit"

97. $choice = Read-Host "Enter your choice"

98. switch ($choice) {

99. "1" { Initialize-GitRepository }

100. "2" { Get-GitStatus }

101. "3" { Upload-GitChanges }

102. "4" {

103. $optionNumber = Read-Host "Enter option number (1 to 4)"

104. Explain-Option -optionNumber $optionNumber

105. }

106. "5" {

107. Write-Host "Exiting the program."

108. exit

109. }

110. default { Write-Host "Invalid choice. Please try again." }

111. }

112. } while ($true)

113. }

114.

115. # Call the main function to start the program

116. Main

# What are the iterations?

1. It first wrote the whole code without any functions. Zero functions. That makes the code less readable and less customizable later. So I have prompted “Can you use functions and restructure the program?”
2. It then wrote the key function of “upload” with nested “ifs”. Even after a few prompts it refused to get of nested ifs”. Final prompt that worked is “Can you exit the function after each command if the command fails or if the user chooses a “no” and get rid of nested if commands”.
3. At the end of these two key prompts the code looks roughly right and structured and good for future refactoring of the code

# Caveats

Although I have tried a few times before this is a super “childish” fray into this idea.

You may want to seriously consider using dedicated programs for something like this. And you must. The opportunities are not only compelling but survival.

Good luck!